Beyond Playing to Win: Diversifying Heuristics for GVGAI

Beyond Playing to Win: Diversifying Heuristics for GVGAI

Cristina Guerrero-Romero
University of Essex
Colchester, UK
cris.guerrero@essex.ac.uk

Annie Louis†
University of Edinburgh
Edinburgh, UK
alouis@inf.ed.ac.uk

Diego Perez-Liebana
University of Essex
Colchester, UK
dperez@essex.ac.uk

***Abstract*—General Video Game Playing (GVGP) algorithms are usually focused on winning and maximizing score but combining different objectives could turn out to be a solution that has not been deeply investigated yet. This paper presents the results obtained when five GVGP agents play a set of games using heuristics with different objectives: maximizing winning, maximizing exploration, maximizing the discovery of the different elements presented in the game (and interactions with them) and maximizing the acquisition of knowledge in order to accurately estimate the outcome of each possible interaction. The results show that the performance of the agents changes depending on the heuristic used. So making use of several agents with different goals (and their pertinent heuristics) could be a feasible approach to follow in GVGP, allowing different behaviors in response to the diverse situations presented in the games.**

I. Introduction

When creating a heuristic or an agent to play a game, the objective is usually clear: to win. There's little discussion about the fact that most efforts in agent Game AI try to either achieve victory or to maximize the game score in the domain being investigated. For instance, when creating drivers for the Car Racing competition [1], the objective is to be the first car that crosses the finish line. If the agent is playing Mario AI [2], the aim is to reach the end of the level alive, avoiding the hazards on the way and maximizing the score.

Building AI for agents that play games like these can be a simple or a complicated task, depending on the approach followed, but one aspect is common: the programmer typically knows how to win the game. They know that it is safer to overtake another driver in certain circumstances, or that collecting coins is a task worth pursuing to maximize game score. These concepts are usually included in the agents as heuristics that allow the algorithm to explore the search space in a more determined way.

However, when building agents for General Video Game Playing (GVGP), these heuristics are less clear, especially if the game that is being played is unknown a priori. In the concrete case of the General Video Game AI (GVGAI) Framework and Competition[1], with a collection of more than 150 games at the time of this writing, the possibilities of writing heuristics that can guide search in all games are greatly reduced. After three years of GVGAI competitions, there hasn't been a single approach that has been able to achieve more than 60% of victories across the different game sets and, in most cases, winners achieve a rate below 50%.

An assorted range of approaches has featured in the contest during this time, from Evolutionary Algorithms (EA) to Monte Carlo Tree Search (MCTS) techniques, including multiple variants and other more straightforward approaches such as Breadth First Search (BFS) or A*. In some cases, several of these approaches are glued together, using a high level switch that determines the algorithm to use, in the form of a meta-heuristic [3]. All these approaches have, quite naturally and understandably, a common objective: to achieve victory.

This work presents a study across algorithms and game-independent heuristics in 20 games of the GVGAI corpus. However, the aim here is to present different types of game-agnostic heuristics that, rather than aiming to win, have different objectives, such as exploring the level, triggering game interactions and gathering knowledge about the dynamics of the game. The ultimate goal is to pave the path for higher level, general algorithms, which will be able to combine different heuristics efficiently in order to improve the chances of not only winning, but also obtaining a better understanding of the game being played. Gaining and using this understanding, when in GVGAI this information is not directly available, could ultimately increase the performance of the general algorithms in terms of winning.

This paper is organized as follows. Section II details previous work on GVGAI controllers and heuristics. Section III describes the search methods used in this work, and is followed by definitions of proposed game heuristics in Section IV. Section V describes the experimental setup and the different ways these heuristics are evaluated, including a discussion about the results obtained.

II. Background

The General Video Game AI Framework (GVGAI) is a port to Java from the original py-vgdl [4], an initial version implemented by Tom Schaul, as a benchmark for planning and learning problems. Games in this framework are described in the Video Game Description Language (VGDL), that allows the definition of 2D arcade games, for single and two-players, where the player controls an avatar that is able to interact with other sprites in the game. The engine exposes a Java

†Work done while at University of Essex
[1]http://www.gvgai.net/

object interface to query the game status (game ticks, score and winning conditions), the state of the avatar (position, velocity, orientation, etc.), the available actions and a view to the other sprites of the game via *Observations*. These, anonymize the type of the sprite by providing an arbitrary integer identifier, but includes its real position in the level grid.

More importantly, the framework provides the AI agent with a forward model, which can be used to foresee the next states reached after an action has been taken in a simulated manner. The controller (agent) must return an action within the 40ms of budget time at each tick, and it counts on 1s for initialization at the beginning of the game. Additionally, the engine does not provide the VGDL description of the game, nor the rules or ways to achieve victory, leaving this to the agent to discover.

GVGAI has run as a competition since 2014, receiving the submission of more than a hundred of entries since. Apart from the single player planning [5], a two-player track [6] spawned in 2016 and attracted eight entries in its first edition of the contest. Both tracks were mostly dominated by tree search techniques, like Adrien Couëtoux's OLETS for single [5] and two-player games [6] (and used in this work) or the winner of the 2015 and 2016 editions, YOLOBOT [7], who combined MCTS, BFS and targeting heuristics.

Heuristics are the base of every general algorithm, and could be decisive for its performance. This paper presents four heuristics, with different goals: *winning*, *exploration*, *knowledge discovery* and *knowledge estimation*. Some authors have looked onto the creation of heuristics of this kind before. Perez et al. [8] developed a domain independent exploration heuristic, based on pheromones that would prevent the agent from staying in the same position too often, hence encouraging exploration of the level. This heuristic was used later in [9], for a Multi-Objective implementation of MCTS (MO-MCTS) in GVGAI, which employed this exploration heuristic as another objective to optimize during play. Interestingly, the authors showed that MO-MCTS was able to improve the performance of the single-objective MCTS algorithm that used a linear combination of the objectives as value function, and also that simply switching between objectives (uniformly at random, one at a time) was able to obtain decent results in many games.

In a similar work to the knowledge discovery and estimation heuristic presented in this paper, Perez et al. [10] proposed an heuristic based on rewarding interactions with sprites in the game to estimate the value of colliding with them. The final algorithm, also hybridized with an Evolutionary Algorithm to guide the Monte Carlo simulations, improved the performance of a vanilla MCTS in the first set of GVGAI games.

Likewise, C. Chu et al. [11] propose the use of potential fields to replace the Euclidean distances employed in [10], by including attractors and repellers in the level based on the influence of the surrounding sprites, slightly improving the victory rate of the algorithms in this set. Additionally, H. Park et al. [12] propose the use of Influence Maps (IM) to help MCTS navigate the search space more efficiently. By means of the forward model, the authors interact with sprites in order to estimate the goodness of all game entities and build an IM, using the concept of *curiosity* as a way to stimulate the agent towards those sprites is has less knowledge of. Then, the IM takes part of MCTS Tree Policy as an extra term, guiding the search and delivering some final promising results.

It is worth mentioning that, although diverting goals from winning and maximizing score is an innovative approach for GVGP, exploring alternative heuristics has been investigated outside the scope of general video game playing. An example is the use of agents for player modeling; like the work of Holmgård et al. in [13] [14], where agents (*Procedural Personas*) were trained to simulate human playing style and decision making process in a dungeon-themed puzzle game.

III. Controllers

As the *GVGAI Framework* is used for the GVGAI Competition, a range of sample general controllers are available. We used five of these sample controllers and adapted them to fit the needs of the experiment.

1) OLETS (Open-Loop Expectimax Tree Search): This agent was created by *Adrien Couëtoux*, winner of the 2014 GVGAI Competition [5]. It has shown a strong performance across different game sets, and it can be easily adapted to different heuristics. The algorithm behind this agent is based on Hierarchical Open-Loop Optimistic Planning (HOLOP), a tree search method that uses sampling to perform search in the game state space. OLETS performs no rollouts and substitutes the average of rewards for a more involved calculation. For more details about this agent, the reader is referred to [5].

2) OLMCTS (Open-Loop Monte-Carlo Tree Search): Version of the Monte-Carlo Tree Search (MCTS) algorithm that uses an open-loop approach [8]. MCTS is a search algorithm that builds an asymmetric tree in memory, which grows towards the most promising parts of the search space. Adding one node to the tree at each iteration, tree navigation is guided by a policy that balances exploration of new actions and exploitation of the most promising ones. Random rollouts are run from the leaves of the tree to estimate the value of the states. For more information on MCTS, its variants and applications, please see [15].

3) OSLA (One Step Look Ahead): One of the simplest algorithms in the sample pool. It checks every available action and estimates the reward gained when each of them is executed, carrying out the one with the best evaluation rating.

4) RHEA (Rolling Horizon Evolutionary Algorithm): RHEA is an online evolutionary algorithm that evolves plans or sequences of actions during the time budget. It has shown a promising performance when compared to MCTS in GVGAI and other domains [16]. Once the best individual is identified, the agent performs the first action in the sequence.

5) RS (Random Search): This algorithm spends the time budget to decide a move in trying random action sequences. As in RHEA, all actions are executed sequentially until end of the plan is reached, where the state is evaluated by an heuristic. The action returned to the game is the first one in the sequence found to maximize this value. RS has recently shown very good results in the same set of games used in this paper [16].

These controllers were taken from the sample pool and modified accordingly in order to provide a common ground for the comparison of both the algorithms and the heuristics used. First of all, whenever possible, the search depth of the algorithms has been set to 10. Note that in some cases (like in OLETS or OSLA), the algorithm itself prevents search from reaching this depth in the time allowed per move. Secondly, the value function that evaluates game states on each algorithm has been isolated so it can be easily exchanged by the different ones presented in this paper. Finally, a form of cumulative reward has been put in place for all algorithms. This allows the longer lookahead algorithms (OLETS, OLMCTS, RHEA & RS) to keep track of the state of the game at every step during the evaluation, as it allows using rewards more accurately during the evaluation process instead of having just the end state of the roll-out/evaluation as a reference. Note that this is already in place by default in OSLA, as it only simulates one step ahead in the search.

IV. Game Heuristics

This section describes the four heuristics explored in this paper. These heuristics affect the way the state is evaluated, and thus guide the search and inform the recommendation policies to play an action in the real game. For each heuristic, we specify its goal, the way the heuristic works, and its pseudocode (Algorithms 1 to 4). As a common ground for all heuristics, it is worth defining the following concepts:

- *Heuristic value*: Result of the evaluation of a game state, as defined in the Algorithms 1 to 4. High and low numeric values for the heuristics are indicated with H^+ and H^-, respectively, and all the other rewards are given in relative terms to these. For the experiments described in this paper, H is set to an arbitrary high value of 10^6.
- *Collision* (same as Interaction or Event): Effect of two sprites being in contact during the game, one of the two sprites being the avatar (i.e. the player). An example could be the player collecting an item in the game.
- *Action-onto*: Particular case of a collision when one of the two sprites in the interaction is a sprite generated by the avatar. An example of this interaction type is a bullet shot by the avatar hitting a target.

A. Winning Maximization (WMH)

Goal: The goal of this heuristic is to win the game.

GVGP algorithms are usually focused on winning the game and, if possible, maximizing the score obtained. This is the aim which all sample agents were following in their original heuristics, even when they were not designed exactly in the same way. *WMH*, as specified in Algorithm 1, penalizes the *end states* where the player loses and rewards those where the agent is proclaimed the winner. In the rest of the states, the difference of the score is set as heuristic value.

Algorithm 1 Winning Maximization (WMH)

```
if is_EndfTheGame() and is_Loser() then
    return H^-
else if is_EndOfTheGame() and is_Winner() then
    return H^+
return new_score - game_score
```

Algorithm 2 Exploration Maximization (EMH)

```
if is_EndfTheGame() then
    return H^-
else if is_outOfBounds(pos) then
    return H^-
if not hasBeenBefore(pos) then
    return H^+/100
else if is_SameAsCurrentPos(pos) then
    return H^-/200
return H^-/400
```

B. Exploration Maximization (EMH)

Goal: The goal of this heuristic is to maximize exploration of the level.

This heuristic is focused on rewarding the agent that visits as many different locations of the level as possible. At the beginning of the game, an empty exploration matrix is initialized. Every time step, this exploration information is updated to annotate the current position of the player as *visited*. This data is used by the heuristic when subsequent states are evaluated, providing a positive reward in those locations where the agent has never been before. In addition, negative rewards are given if the agent remains in the same position in consecutive moves. Algorithm 2 shows the pseudocode for this heuristic.

C. Knowledge Discovery (KDH)

Goal: The goal of this heuristic is to interact with the game as much as possible, triggering sprite spawns and interactions.

This heuristic is focused on maximizing the discovery of the different sprites present in the game, and trying to perform interactions with all of them. We say that the agent has *acknowledged* a sprite if its type has been observed either in gameplay or in the simulations using the forward model. At the beginning of the game, the *acknowledgement* of the sprites is initialized using the available information (i.e. what sprites are visible right before the game starts). Regarding the interactions with sprites, the following types have been considered:

- *Collision*. When the agent itself bumps into another sprite of the game.
- *Action-onto*. When an item previously created by the agent (typically because an action of type *ACTION* was executed previously) has collided with another sprite of the game.

Every interaction with another sprite is recorded in an interaction table. *KDH* rewards those states where new sprites are acknowledged. If no new sprites emerge in the next state reached, it prioritizes carrying out new interactions. Ultimately, if no new interactions are at their disposal either, the heuristic rewards interactions that have occurred in the past, but if they happen now in different locations of the level.

Algorithm 3 Knowledge Discovery (KDH)

if $is\_EndfTheGame()$ **and** $is\_Loser()$ **then**
  return H^-
else if $is\_EndfTheGame()$ **and** $is\_Winner()$ **then**
  return $H^-/2$
else if $is\_outfBounds(pos)$ **then**
  return H^-
if $newSpriteAck()$ **then**
  return H^+
if $eventOccured(lastTick)$ **then**
  if $is\_newUniqueInteraction(event)$ **then**
    return $H^+/10$
  else if $is\_newCuriosityCollision(event)$ **then**
    return $H^+/200$
  else if $is\_newCuriosityAction(event)$ **then**
    return $H^+/400$
return $\frac{H^-}{400}$

Algorithm 4 Knowledge Estimation (KEH)

if $is\_EndfTheGame()$ **and** $is\_Loser()$ **then**
  return H^-
else if $is\_EndfTheGame()$ **and** $is\_Winner()$ **then**
  return $H^-/2$
else if $is\_outfBounds(pos)$ **then**
  return H^-
if $newSpriteAck()$ **then**
  return H^+
if $eventOccured(lastTick)$ **then**
  if $is\_newUniqueInteraction(events)$ **then**
    return $H^+/10$
  return $rewardForTheEvents(events)$ $\{\in [0, \frac{H^+}{100}]\}$
$n\_int = getTotalNStypeInteractions(int\_history)$
if $n\_int == 0$ **then**
  return 0
return $H^-/(200 \times n\_int)$ $\{\in [\frac{H^-}{200}, 0]\}$

This is referred to as *curiosity* in the rest of this paper. Please refer to Algorithm 3 for the pseudocode of this heuristic.

D. Knowledge Estimation (KEH)

***Goal**:* The goal of this heuristic is to predict the outcome of interacting with sprites, both changes in the victory status and in score modifiers.

This heuristic is focused on acquiring the best possible knowledge of the game dynamics, in order to estimate the advantages and disadvantages of each possible interaction. The goal is to provide an estimation of the winning/losing conditions and the score change when interacting with each one of the sprites present in the game. During the game, the heuristic gathers the following information for each interaction type (*collisions* and *actions-onto*) with every sprite:

- *Win condition*. Depending on the game, interactions between sprites can trigger a termination condition or not. After an interaction has been detected, the game could have finished because that particular interaction meets a termination requirement, so the total number of wins and defeats encountered are collected. This information will be used to predict (by simply calculating the average) the win condition at the end of the game, which ideally would be 0 (if that specific interaction never produces a game over), or 1 or -1, in case it triggers a termination condition making the player to win or lose, respectively.
- *Score change*. Some interactions trigger a modification in the score of the game, which is the second piece of information that *KEH* tries to predict. Therefore, the change in the score presumably derived from a detected interaction is accumulated. At the end of the game, this accumulated information will be used to estimate the score change per interaction (again, by calculating the average of score change for this sprite), which ideally will be 0 if the specific interaction does not affect the score at all or a value (positive or negative) if it does.

As it is possible that some sprites in the game have never been subject of an interaction by the avatar, a default value of 0 is given for both Win Condition and Score Change. Effectively, this implies an assumption that, upon lack of interactions with a given sprite, it will predict that no score or changes on game termination are triggered by this sprite. Therefore, but rather than maximizing the number of interactions with the same sprites, *KEH* attempts to uniformly interact with all the available ones, in order to better estimate the effects of these collisions and improve upon the default estimations. Algorithm 4 indicates the pseudocode of this heuristic.

V. Experimental Work

A. Experimental Setup

There are more than a hundred of games available in the GVGAI Framework, all of them with different properties and characteristics. As testing the methods proposed in this paper in all games in the framework is prohibitively expensive, we selected a subset of them in a way that best represents the variety of games in the benchmark. Some authors had already looked into a way of selecting games with the aim of using the GVGAI framework as a benchmark for experiments, for instance the work of Gaina et al [16], [17] on Rolling Horizon evolutionary methods.

On account of this, the authors combined two classifications already presented in previous works: a selection of games based on how Monte Carlo Tree Search (MCTS) performed in them [18], and a classification of games in clusters by their features [19]. The final set is constituted by 20 games, presented in Table I, and it is the one employed in this paper.

Five agents (OLETS, OLMCTS, OSLA, RHEA and RS) have been run with four different heuristics (WMH, EMH, KDH and KEH), for a total of 20 different configurations used in this experiment. Each of these agents have played the first level of each game for 20 times, to reach a total of 400 games played for each agent with an heuristic. For each

Deterministic games		Stochastic games	
Bait	*Camel Race*	*Aliens*	*Butterflies*
Chase	*Escape*	*Chopper*	*Crossfire*
Hungry Birds	*Lemmings*	*Digdug*	*Infection*
Missile Command	*Modality*	*Intersection*	*Roguelike*
Plaque Attack	*Wait For Breakfast*	*Seaquest*	*Survive Zombies*

TABLE I
NAMES OF THE 20 GAMES FROM THE SUBSET SELECTED [17].

WMH Stats		
Controller	**F-1 Points**	**Total average % of Wins**
OLETS	**449**	**59.00** (5.43)
RS	356	51.00 (4.24)
OLMCTS	333	41.50 (3.69)
OSLA	283	34.00 (4.95)
RHEA	224	10.00 (3.29)

TABLE II
WMH STATS TABLE, PRESENTING THE **OVERALL AVERAGE** OF PERCENTAGE OF WINS OBTAINED FOR EACH OF THE CONTROLLERS

heuristic, controllers are sorted according to their results in each game, based on a criteria tailored to the heuristic employed. Therefore, this section presents four different rankings, one per heuristic, determining the best algorithms at each task.

In all rankings, the Formula 1 point system used in the GVGAI Competition has been used to sort the agents. As five algorithms are used in this experiment, they receive 25, 18, 15, 12 or 10 points, depending on the position achieved. The final ranking for the heuristic processed is determined by the total sum of the scores received across the 20 games. This point system has been previously adopted by authors using the GVGAI framework for performance benchmarking [20] as it allows a fair perspective on the general performance of the agents when a set of games is considered.

Apart from the final rankings for each of the heuristics, some statistics are included in order to provide an overview of the overall performance of the agents. Note that some of the criteria in the ranking benchmark is used merely for breaking ties and are not really worth to be summarized (i.e. game ticks results). Some other data is highly dependent on the games and cannot be generalized or summarized in a unique table per heuristic (i.e. score for WMH) and it has not been included in the statistics unless it has been found (although game-relative still) generalizable enough to be comparable.

B. Rankings for WMH

The data gathered at the end of the game for the Winning Maximization Heuristic includes three values: win condition (a 1 or a 0, determined by the game finishing with a victory for the agent or not, respectively), score (number of points at the end of the game) and timesteps (game ticks played).

Given the results of different controllers in a game, the agent with the highest percentage of victories is considered the best one. In case of a tie, higher scores are better. For game ticks, a lower average of timesteps to victory is preferred (indicative that the game was won faster), while a higher value is better for games lost (which suggest a longer survival time). This ranking system is similar to the one used in the GVGAI competition.

Table II shows the final rankings according to this system. The statistics included in this case are the overall average of percentage of victories obtained for each of the controllers.

The performance of *RHEA* is noticeably poor compared with the rest of the algorithms, being last in the ranking with a mere 10% (3.29) overall percentage of victories, in contrast with the 34.00% (4.95) obtained by *OSLA* (ranked in 4^{th} position) and the 59.00% (5.43) achieved by *OLETS*, ranked 1^{st} and with the best stats for the WMH. Unlike *RHEA*, *RS* performs well, finishing in 2^{nd} position of the ranking with the second best numbers. It is interesting that these two algorithms, even while following a similar approach, have clearly different performance, indicating that further tweaking of the default *RHEA* parameters could land better results. It is worth mentioning that, specifically for the game *Intersection*, the difference between this algorithm and the other ones is notable, as RHEA averages no victories when for the rest the average is 100.00%.

Result tables with performance on a game per game basis have not been included in this paper for the sake of space, but some interesting results have been observed and are discussed next. Firstly, an average of victories of 0.00% has been found for *CamelRace*, *Digdug*, *Lemmings* and *Roguelike*, meaning that none of the controllers managed to win these games. The last three corresponds with games with big levels where the agent must carry out accurate actions in order to trigger the winning condition and, because of the limited resources employed in this heuristic, is understandable that any of them managed to find the solution. However, it is interesting that the first game has not been solved, given that to win *Camel Race*, moving in a straight line to the right, where the goal is located, is enough. Clearly, even the simplest of the games poses a challenge for agents when the information about the dynamics of the game is restricted. Also *Hungry Birds* was very close to fall onto the category of unsolvable games as only *OLETS* managed to win it, with a prominent 65% (10.67) average of victories. Similarly, *Crossfire* was only solved by three of the algorithms (OLETS, OSLA and RS) with a very low percentage of victories achieved by the last two (5.00% (4.87)). Regarding games with an overall high percentage of victories (higher than 80% on average), we could mention *Aliens*, *Infection*, *Intersection* and *Modality*. In summary, it can be concluded that *OLETS* works really well with the WMH, while *RHEA* shows a poorer performance.

C. Rankings for EMH

The rankings designed for the Exploration Maximization Heuristic consider two main aspects. First, the number of different positions marked as visited in the heuristic's exploration matrix is used to calculate the percentage of the level explored. As the number of valid positions depends on the game and level played, and that information is not available through the agent's interface, the maximum exploration was calculated by hand for all levels. On a given game, a higher exploration

EMH Stats		
Controller	**F1-Points**	**Total average % Explored**
RS	**428**	74.94 (1.83)
OLETS	377	**76.86** (2.19)
OLMCTS	309	65.60 (1.64)
OSLA	282	54.14 (2.18)
RHEA	204	27.56 (1.64)

TABLE III
EMH STATS TABLE, PRESENTING THE **OVERALL AVERAGE** OF PERCENTAGE EXPLORED OBTAINED FOR EACH OF THE CONTROLLERS

KDH Stats					
Controller	**F1-Points**	**% Ack (Rel)**	**% Int (Rel)**	**% CC (Rel)**	**% CA (Rel)**
RS	**414**	**100.00**	**96.18**	85.46	**87.42**
RHEA	342	99.66	95.48	62.48	54.44
OLMCTS	330	99.79	93.53	84.75	84.06
OLETS	279	99.86	88.97	**90.72**	77.55
OSLA	235	98.48	84.99	56.37	51.75

TABLE IV
KDH RANKINGS AND **OVERALL GAME-RELATIVE** (REL) PERCENTAGES FOR SPRITES ACKNOWLEDGED (ACK), INTERACTIONS ACHIEVED (INT), CURIOSITY COLLISION (CC) AND CURIOSITY ACTION-ONTO (CA)

percentage is better. In order to break ties, the game tick when the agent visited the last new position is recorded, being the smaller the better in order to reward a faster exploration.

Table III shows the final rankings obtained for this heuristic. The ranking obtained for the Exploration Maximization Heuristic looks very similar to the one for WMH. Interestingly, the overall average of percentage of exploration for RS and OLETS is almost the same, and even slightly higher for OLETS (74.94% (1.83) and 76.86% (2.19)), but it is *RS* the algorithm achieving the 1^{st} position, with a total of 428 points, being *OLETS* second with 377. OLMCTS is third with 309 points and a performance of 65.60% (1.64). RHEA has a notably poor performance compared with the rest of the algorithms (27.56% (1.64)) and ranks last, headed by OSLA, in 4^{th} place, with a performance of 54.14% (2.18).

For the per-game analysis, only in *Aliens* some of the agents (RS, OLETS and OLMCTS) manage to achieve a 100.00% of exploration, but games like *Butterflies*, *Chase*, *Chopper*, *Modality* and *Survive Zombies* present a high performance from the agents, with a total average of more than 80%. In all these games, the level played is completely accessible, meaning that the agent does not have to make an action or interact with other sprites in order to be able to reach the positions in the navigation matrix. Only two of the games have a significant mediocre average performance of the controllers: *Camel Race*, which finishes very quickly (as the NPC win the game rapidly) and does not provide the agents enough time to explore; and *Roguelike*, which has a large map with two distinguished zones; the agents having to collect a key in order to access to the second one.

D. Rankings for KDH

For the Knowledge Discovery Heuristic, agents are ranked according to the following criteria (each point is a tie-breaker of the previous one):

- *Sprites acknowledged*: Number of different game entities acknowledged. This is, how many different sprites type IDs are observed during the game or the forward model simulations. The higher the better.
- *Unique interactions*: Number of collisions and actions-onto involving different pair of sprites, the higher the better.
- *Curiosity*: Number of interactions with sprites in different locations of the game, favoring higher values for collisions than action-onto curiosity instances. Note that both positions and sprites are taken into account, making more than one curiosity interaction possible in a certain location if it involves different sprites.
- *Last acknowledgment game tick*. Game tick of the last new sprite acknowledged. As in the next two points, the lower this value, the better.
- *Last interactions game ticks*. Game tick of the last new interaction that took place.
- *Last curiosity game ticks*. Game tick of the last new curiosity instance recorded (either for collision or for the action-onto type).

Table IV shows the final rankings obtained for this heuristic. Here, the statistics are given in percentages, in contrast with the criteria used for the F1 ranking. As there is no information about the total number of sprites acknowledgeable per game, and it is not possible to gather this data accurately, a benchmark based on relative information per game (number of sprites) was used. However, this measurement could not be generalized to provide valid overview statistics, as the number of sprites is different between games. Therefore, another approach was followed. For each game, the highest value obtained for any of the controllers was used as benchmark to calculate the relative performance percentage for each of the agents for that particular game; meaning that for every game there is always at least one algorithm with a performance of 100%. The statistics displayed in the table give the average of this percentage performance through every game.

For KDH, RS ends up in 1^{st} position with 414 points, followed by RHEA with 342 points, OLMCTS with 330 points, OLETS with 279 points and OSLA last with 235 points. RS is the algorithm that performs the best, achieving 100% performance in acknowledgement, meaning that it is the only controller that was capable of acknowledging the maximum number of sprites for every game. Also, for the other achievements considered, it always performs the first or the second with high performance, which is a remarkable result. The algorithm that performs the worst in general is OSLA as, although its performance acknowledging the elements and interacting with them it is over 80%, the curiosity performance is just around a 50%. The best performance for *collision curiosity* is achieved by OLETS' with a 90.72%.

E. Rankings for KEH

The data collected for the Knowledge Estimation Heuristic rankings must reflect how well the agent is able to estimate

KEH Stats			
Controller	**F1-Points**	**Avg. Square Error Average**	**% Int estimated**
OLMCTS	**347**	**0.338**	97.92
RHEA	330	0.505	97.50
OSLA	313	0.617	73.19
RS	310	0.528	**98.33**
OLETS	300	1.086	87.92

TABLE V
KEH RANKINGS AND **OVERALL AVERAGE** OF THE SQUARE ERROR AVERAGE AND THE **OVERALL GAME-RELATIVE** (REL) PERCENTAGE FOR THE INTERACTIONS ESTIMATED (INT ESTIMATED)

Rankings								
	WMH		**EMH**		**KDH**		**KEH**	
1^{st}	**449**	**OLETS**	**428**	**RS**	**414**	**RS**	**347**	**OLMCTS**
2^{nd}	356	RS	377	OLETS	342	RHEA	330	RHEA
3^{rd}	333	OLMCTS	309	OLMCTS	330	OLMCTS	313	OSLA
4^{th}	283	OSLA	282	OSLA	279	OLETS	310	RS
5^{th}	224	RHEA	204	RHEA	235	OSLA	300	OLETS

TABLE VI
AGENTS RANKED BY HEURISTIC, INCLUDING THE TOTAL NUMBER OF POINTS ACHIEVED BY EACH ALGORITHM.

the dynamics of the game, in terms of score awarded when colliding with a sprite, and changes on the victory status. In order to achieve this goal, estimations of the outcomes of each interaction type (*collisions* and *actions-onto*) are gathered from the agent when the game is over, in terms of the likelihood of winning or losing the game and the score awarded after colliding with such sprite. These estimations must be compared with the true outcome in order to determine how accurate these predictions are. The ground truth about each one of the games has been extracted manually, regarding sprites that cause score change or winning/losing the game.

For every prediction, the square error to the ground truth is calculated, and the mean of square errors determine the total prediction error incurred by the agent in that game. This quantity is the first decisive factor for the rankings, the best controller being the one with the smallest average of square errors. In the (rare) case of a tie, the number of interactions that the controller was able to give a prediction for, is used as a tie-breaker, considering the higher the better.

Table V shows the rankings obtained with this criteria, including the average of square errors in all games. The statistics presented for KEH are the overall average of the average of square errors obtained for each the games and the percentage of interactions estimated. Notice that this interactions percentage has been obtained game-relatively, considering the highest value given by a controller for each of the games as benchmark to obtain the performance. The value displayed in the table has been obtained with the average of these values for all games.

OLMCTS ranks first for KEH with 347 points, followed by RHEA with 330 points. Last three positions, OSLA, RS and OLETS are very close to each other in number of points achieved: 313, 310 and 300, respectively. It is noticeable how the average square error is not very good overall, as the best performance average is OLMCTS's 0.338. Also, none of the agents has remarkably better performance than the others, as the difference of points between OLMCTS and OLETS is just 47. However, it is worth mentioning that, unless there is a poor overall performance, the estimations for some of the interactions in certain games are very accurate. There are even a few cases where the outcome of the interaction with a determined sprite is estimated by all the controllers with an average square error of 0.00%. Most of these cases are related with estimations for the interactions of the type *collision*.

In *Aliens*, all five controllers are capable of giving an accurate estimation for the collisions with both the *bomb* and the *alien* sprites, predicting an inarguable defeat. Another example of accuracy has been found in *Chase*, where exists a sprite of type *'angry goat'* that can (or not) emerge at some point during the game and kills the avatar when colliding with it, also removing one point from the score. OLMCTS, RS and RHEA managed to discover this sprite and interaction, and predicting both winning condition and score change with an average square error of 0.00%. Finally, it is worth mentioning how RHEA is capable of predicting with an average square error of 0.00% every outcome of every sprite interaction for the game *Escape*, which is a remarkable achievement. These are only some of the results of good performance detected when analyzing the data obtained, but many examples with a bad performance have been encountered. It shows how challenging and difficult is the task of predicting and trying to understand the game when its information is very limited.

F. Rankings overview

Table VI provides an overview of the rankings obtained for each of the heuristics. It displays the position of each of the controllers and the total number of points achieved. The number of points scored by the controllers in the first positions for WMH, EMH, KDH and KEH are, respectively, 449, 428, 414 and 347. Note that the difference between the first and last positions for WMH, EMH and KDH is higher than for KEH, with a more uniform distribution of points in the last case, suggesting that the performance of the algorithms for KEH is very similar, with no algorithm showing a clear dominance.

It is worth mentioning the heterogeneous results obtained, as three different controllers (OLETS, RS and OLMCTS) have reached a first position and other three (RHEA, OSLA and OLETS) have lost in at least one of the rankings. In addition, there is a noticeable difference for each of the heuristics: RHEA performs poorly for both WMH and EMH (being last and with a significant difference in scores with OSLA, in 4^{th} place) but appears on the top of the ranking for KDH and KEH, ranking 2^{nd} in both of them with 342 and 330 points. There is also a remarkable change in the order of the controllers; unlike for RHEA, OLETS performs very well for WMH and EMH but ranks 4^{th} and last in both of the heuristics involving knowledge. OSLA maintains a similar rank through over all the heuristics (4^{th} for WMH and EMH, last for KDH and 3^{rd} for KEH) and OLMCTS, with a medium performance for WMH, EMH and KDH, reaches the first position for KEH. Finally, it can be said that RS has generally good performance,

being second for WMH, first for EMH and KDH and, even in a fourth position for KEH, the difference of points between the positions in the rank is not as high as in the other rankings.

VI. Conclusions

The main purpose of the experiment conducted in this work was to analyze how known general agents perform when their objective is changed. By means of different heuristics, the goal of the search methods is modified to explore, interact or predict, rather than wining games. 5 different sample GVGAI controllers are employed for this study (OLETS, RS, OLMCTS, OSLA and RHEA), and 4 heuristics were designed: *winning*, *exploration*, *knowledge discovery* and *estimation*.

This work can be taken as a first step in the possibility of enlarging GVGP techniques. These would use and combine different heuristics to gain useful knowledge about the dynamics of the game and improve the performance of the general algorithms. It is worth mentioning that, even when the GVGAI framework has been used for the experiment, these strategies should be applicable to GVGP in general, and even extended to put into practice a general evaluation of levels and games.

From the heterogeneous results, we can deduce two important things: how challenging and difficult is the task of achieving different goals with a good performance for every game when it is generalized; and how the performance of the agents changes depending on the heuristic used, a very interesting and thought-inspiring result. It is sensible to think that all the heuristics presented here may come in handy at different moments of playing a game; for instance, it sounds reasonable that a first stage of play would use the *knowledge discovery* and *knowledge estimation* heuristics to achieve a better understanding of the game (note that all heuristics specifically penalize losing the game - thus this can be considered a "safe" search). Once certain conditions have been met, it could be the time to proceed to use a combination of *winning* and *exploration* heuristics (for instance, in a multi-objective setting, as in [9]) to try to achieve victory and maximize score, but influenced for the discoveries found in the previous stage. Furthermore, it could be possible to design a high level meta-heuristic algorithm, capable of combining and selecting different agents and heuristics in-game. Having an available set of general algorithms with different objectives (provided by the pertinent heuristic) could allow the agent to accommodate to different situations that emerge during the games, and to switch behavior in response to the environment.

In terms of using the right controller, it would be reasonable to choose the one with best results for the heuristic to be used; or the one with steady results, if several heuristics are brought together. If combining WMH and EMH, the choice would be between using RS or OLETS but, if knowledge is also included, it would be preferable to use RS or OLMCTS. Future work will combine the idea of exploring and exploiting the level, making the most from the knowledge acquired in order to improve the performance in terms of winning.

Last but not least, although there is still room for improvement, these agents and heuristics are capable of obtaining a relatively good understanding of the game being played. This information can be used not only to play games better, but also to aid general procedural content generation of levels and games, clearly being an alternative for generators for the Level and Rule Generation GVGAI competition [7].

Acknowledgment

This work was funded by the EPSRC CDT in Intelligent Games and Game Intelligence (IGGI) EP/L015846/1.

References

[1] D. Loiacono *et al.*, "The 2009 Simulated Car Racing Championship," *IEEE Trans. on Computational Intelligence and AI in Games*, vol. 2:2, pp. 131–147, 2010.

[2] J. Togelius, N. Shaker, S. Karakovskiy, and G. N. Yannakakis, "The Mario AI Championship 2009-2012," *AI Magazine*, vol. 34, no. 3, pp. 89–92, 2013.

[3] A. Mendes, A. Nealen, and J. Togelius, "Hyperheuristic General Video Game Playing," *Proceedings of Computational Intelligence and Games (CIG). IEEE*, 2016.

[4] T. Schaul, "A Video Game Description Language for Model-Based or Interactive Learning," in *Conference on Computational Intelligence in Games (CIG)*. IEEE, 2013, pp. 1–8.

[5] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 General Video Game Playing Competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, 2016.

[6] R. D. Gaina, D. Perez-Liebana, and S. M. Lucas, "General Video Game for 2 Players: Framework and Competition," in *Proc. of the IEEE Computer Science and Electronic Engineering Conference*, 2016.

[7] D. Perez-Liebana, S. Samothrakis, J. Togelius, S. M. Lucas, and T. Schaul, "General Video Game AI: Competition, Challenges and Opportunities," in *13th AAAI Conference on Artificial Intelligence*, 2016.

[8] D. Perez Liebana, J. Dieskau, M. Hunermund, S. Mostaghim, and S. Lucas, "Open Loop Search for General Video Game Playing," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2015, pp. 337–344.

[9] D. Perez-Liebana, S. Mostaghim, and S. M. Lucas, "Multi-Objective Tree Search Approaches for General Video Game Playing," in *IEEE Congress on Evolutionary Computation (CEC)*, 2016, pp. 624–631.

[10] D. Perez, S. Samothrakis, and S. Lucas, "Knowledge-Based Fast Evolutionary MCTS for General Video Game Playing," in *Conference on Computational Intelligence and Games*. IEEE, 2014, pp. 1–8.

[11] C. Y. Chu, T. Harada, and R. Thawonmas, "Biasing Monte-Carlo Rollouts with Potential Field in General Video Game Playing," , 2015.

[12] H. Park and K.-J. Kim, "MCTS with Influence Map for General Video Game Playing," in *Conference on Computational Intelligence and Games (CIG)*. IEEE, 2015, pp. 534–535.

[13] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Generative Agents for Player Decision Modeling in Games." in *FDG*, 2014.

[14] ——, "Evolving Personas for Player Decision Modeling," in *Computational Intelligence and Games (CIG)*. IEEE, 2014, pp. 1–8.

[15] C. B. Browne *et al.*, "A Survey of Monte Carlo Tree Search Methods," *IEEE Trans. on Computational Intelligence and AI in games*, vol. 4:1, pp. 1–43, 2012.

[16] R. D. Gaina, J. Liu, S. M. Lucas, and D. Pérez-Liébana, "Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2017, pp. 418–434.

[17] R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, "Population Seeding Techniques for Rolling Horizon Evolution in General Video Game Playing," in *Proc. of the Congress on Evolutionary Computation*, 2017.

[18] M. J. Nelson, "Investigating Vanilla MCTS Scaling on the GVG-AI Game Corpus," in *Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 1–7.

[19] P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius, "Matching Games and Algorithms for General Video Game Playing," in *12th Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[20] D. Pérez-Liébana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "Analyzing the Robustness of General Video Game Playing Agents," in *Conference on Computational Intelligence and Games*, 2016.